Advanced Encryption Standard (AES)

Dr. Bhaskar Mondal

Advanced Encryption Standard AES

"It seems very simple."

"It is very simple. But if you don't know what the key is it's virtually indecipherable."

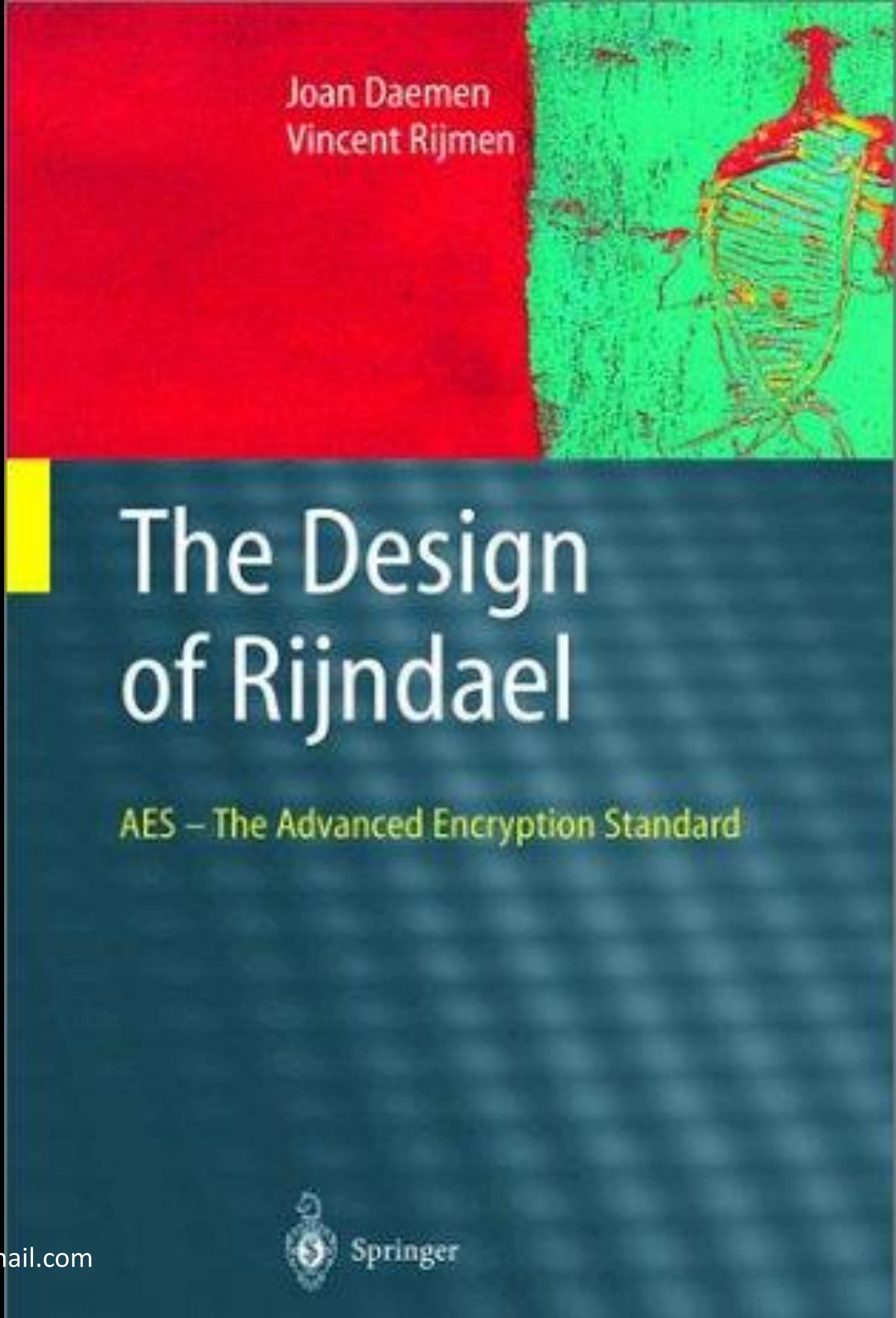
—*Talking to Strange Men,* Ruth Rendell

Advanced Encryption Standard

- In 1997, NIST made a formal call for algorithms stipulating that the AES would specify an unclassified, publicly disclosed encryption algorithm, available royalty-free, worldwide.
- Goal: replace DES for both government and private-sector encryption.
- The algorithm must implement symmetric key cryptography as a block cipher and (at a minimum) support block sizes of 128-bits and key sizes of 128-, 192-, and 256-bits.
- In 1998, NIST selected 15 AES candidate algorithms.
- On October 2, 2000, NIST selected **Rijndael** (invented by Joan Daemen and Vincent Rijmen) to as the AES.

AES Features

- Designed to be efficient in both hardware and software across a variety of platforms.
- Not a Feistel Network
- Block size: 128 bits
- Variable key size: **128, 192, or 256 bits.**
- Variable number of rounds (10, 12, 14):
  - 10 if K = 128 bits
  - 12 if K = 192 bits
  - 14 if K = 256 bits
- No known weaknesses



Origins

- a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow with small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were short-listed in Aug-99
- Rijndael was selected as the AES in Oct-2000
- issued as *Federal Information Processing Standards (FIPS)* PUB 197 standard in Nov-2001

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- initial criteria:
  - security – effort to practically cryptanalyse
  - cost – computational
  - algorithm & implementation characteristics
- final criteria
  - general security
  - software & hardware implementation ease
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

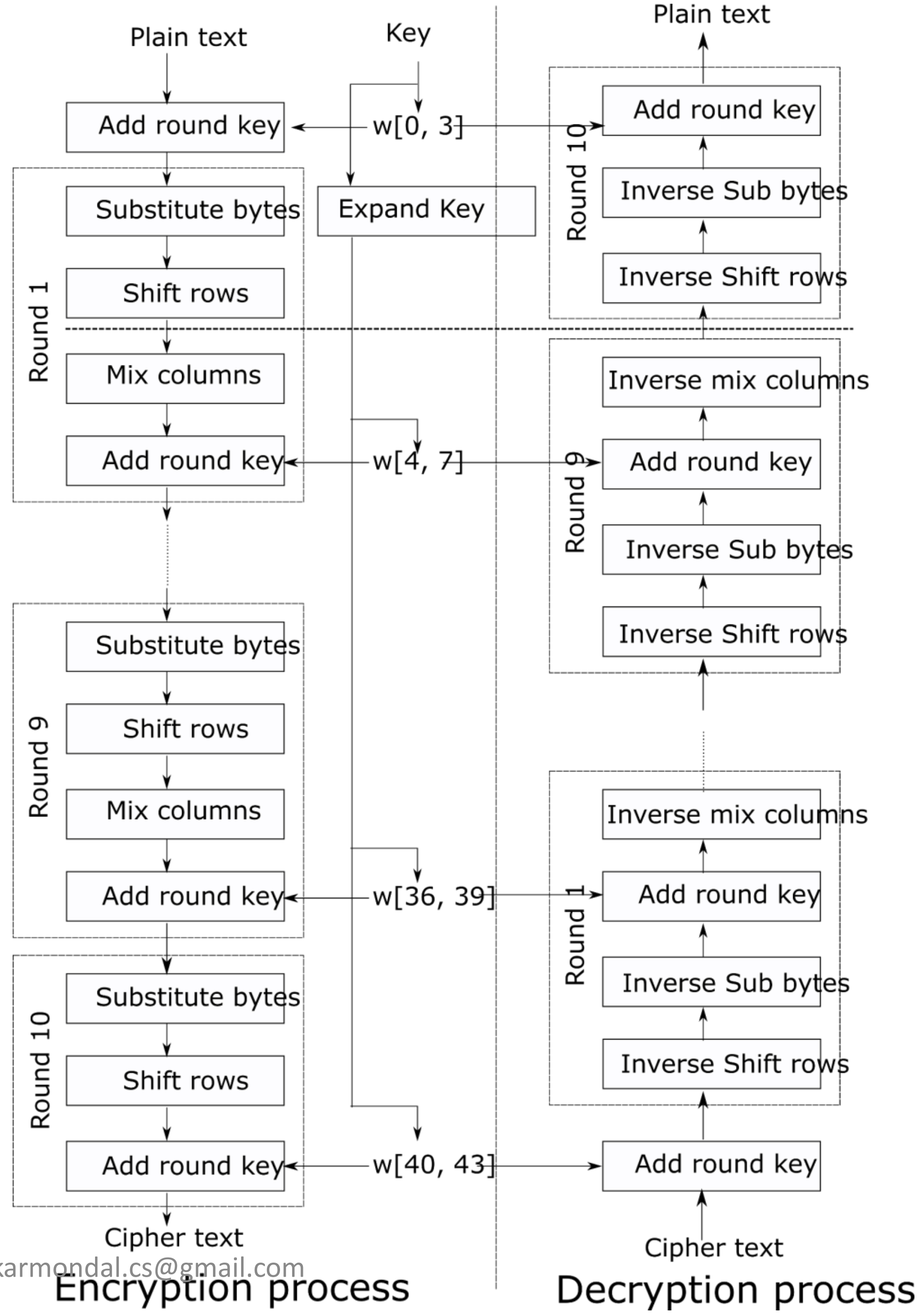
The AES Cipher - Rijndael

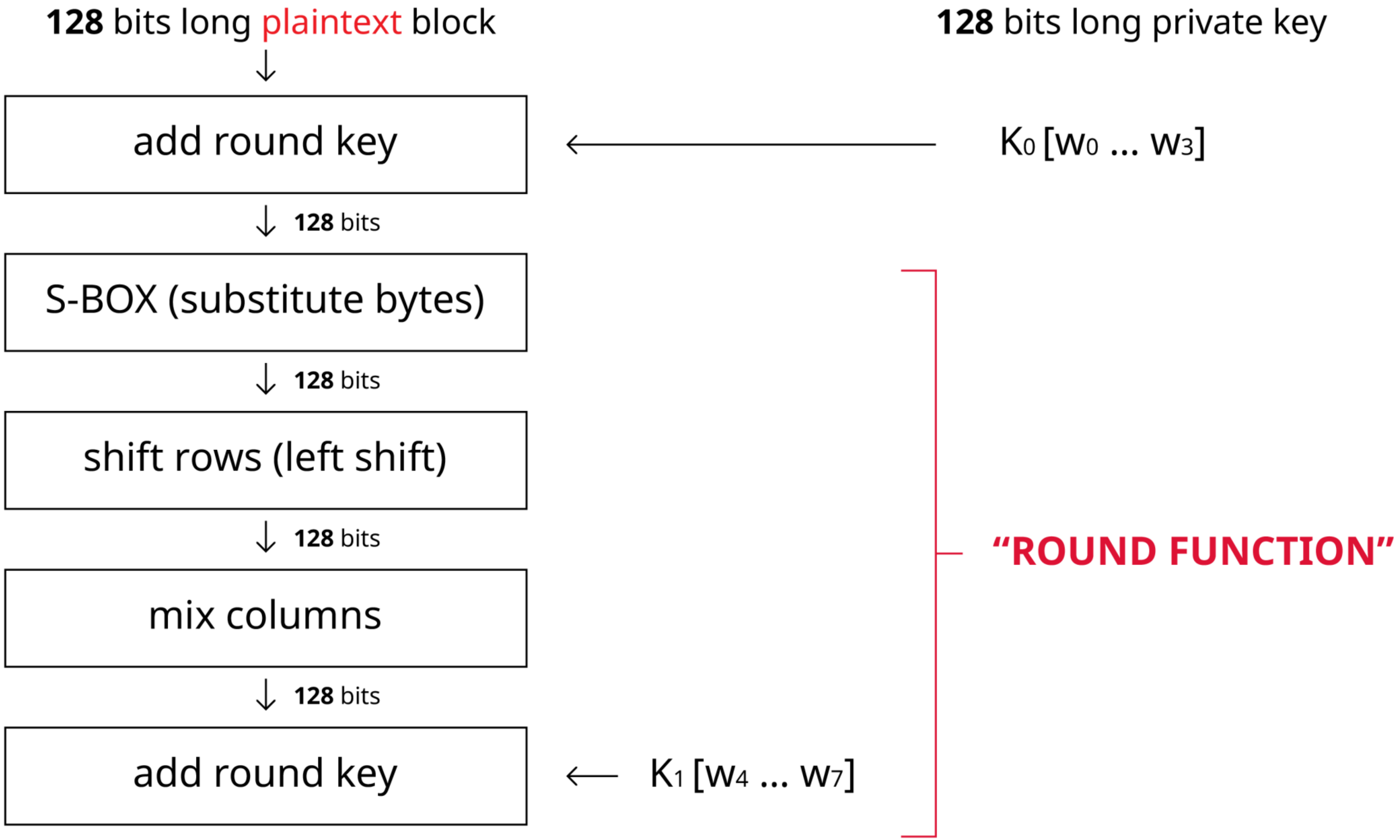
- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- 10 rounds for 128-bit key length, 12 rounds for 192-bit key length, 14 rounds for 256-bit key length
- an **iterative** rather than **feistel** cipher
  - treats data in 4 groups of 4 bytes
  - operates an entire block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Rijndael

- processes data as 4 groups of 4 bytes (state)
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)
- initial XOR key material & incomplete last round
- all operations can be combined into XOR and table lookups - hence very fast & efficient

Rijndael


Plain text
Key
Add round key
w[0, 3]
Substitute bytes
Expand Key
Shift rows
Round 1
Mix columns
Add round key
w[4, 7]
Substitute bytes
Shift rows
Round 9
Mix columns
Add round key
w[36, 39]
Substitute bytes
Round 10
Shift rows
Add round key
w[40, 43]
Cipher text
Encryption process
Plain text
Add round key
Round 10
Inverse Sub bytes
Inverse Shift rows
Inverse mix columns
Add round key
Round 9
Inverse Sub bytes
Inverse Shift rows
Inverse mix columns
Add round key
Round 1
Inverse Sub bytes
Inverse Shift rows
Add round key
Cipher text
Decryption process

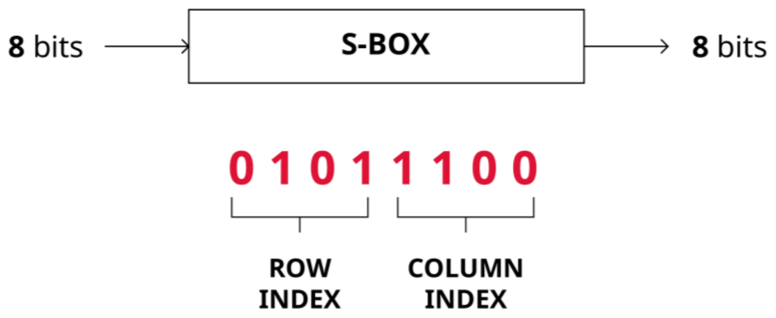

128 bits long plaintext block
128 bits long private key
add round key
K_0 [w_0 ... w_3]
128 bits
S-BOX (substitute bytes)
128 bits
shift rows (left shift)
128 bits
mix columns
128 bits
add round key
K_1 [w_4 ... w_7]
"ROUND FUNCTION"

We represent the data (plaintext, ciphertext and key) as metrixes

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16



Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte in row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by row 9 col 5 byte
  - which is the value {2A}
- S-box is constructed using a defined transformation of the values in GF(2^8)
- designed to be resistant to all known attacks

Shift Rows

a circular byte shift in each row

- 1st row is unchanged
- 2nd row does 1 byte circular shift to left
- 3rd row does 2 byte circular shift to left
- 4th row does 3 byte circular shift to left

decrypt does shifts to right

since state is processed by columns, this step permutes bytes between the columns

Shift Rows

S_0	S_4	S_8	S_{12}	⟵ circular left shift with 0 step
S_1	S_5	S_9	S_{13}	⟵ circular left shift with 1 steps
S_2	S_6	S_{10}	S_{14}	⟵ circular left shift with 2 steps
S_3	S_7	S_{11}	S_{15}	⟵ circular left shift with 3 steps

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

⟶

S_0	S_4	S_8	S_{12}
S_5	S_9	S_{13}	S_1
S_{10}	S_{14}	S_2	S_6
S	S_3	S_7	S_{11}

Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly $m(x) = x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Mix Columns

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix}$$

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \longrightarrow \begin{bmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \\ s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \end{bmatrix}$$

Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption is identical since XOR is own inverse, just with correct round key
- designed to be as simple as possible

1b	22	cb	03
7c	ae	f4	ba
14	01	1b	4f
09	a6	88	4a

…

subkey generation

private key represented as two-dimensional array, and each block has 1byte.

1b	22	cb	03									
7c	ae	f4	ba							…		
14	01	1b	4f									
09	a6	88	4a									

ba
4f
4a
03

subkey generation

First take the right-most column, and execute circular upward shift

Dr. Bhaskar Mondal, NIT Patna, bhaskarmondal.cs@gmail.com

K_{i-4}			K_{i-1}	K_i								
1b	22	cb	03									
7c	ae	f4	ba							...		
14	01	1b	4f									
09	a6	88	4a									

$$\begin{bmatrix} 1b \\ 7c \\ 14 \\ 09 \end{bmatrix} \textbf{ XOR } \begin{bmatrix} f4 \\ 84 \\ d6 \\ 7b \end{bmatrix} \textbf{ XOR } \begin{bmatrix} 01 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} 03 \\ ab \\ 4c \\ a5 \end{bmatrix}$$

subkey generation

XOR operation with K_(i-4) columns and take the predefined value from rcon table, and do XOR operation again. The result first column of current round subkey.

	K_{i-4}			K_{i-1}	K_i							
1b	22	cb	03	03						…		
7c	ae	f4	ba	ab								
14	01	1b	4f	4c								
09	a6	88	4a	a5								

$$\begin{bmatrix} 22 \\ ae \\ 01 \\ a6 \end{bmatrix} \ \textbf{XOR} \ \begin{bmatrix} 03 \\ ab \\ 4c \\ a5 \end{bmatrix} = \begin{bmatrix} 01 \\ 22 \\ a3 \\ 88 \end{bmatrix}$$

subkey generation

Generating 2nd, 3rd and last column of subkey is rather simple, just do XOR operation on K_(i-1) and K_(i-4) column.

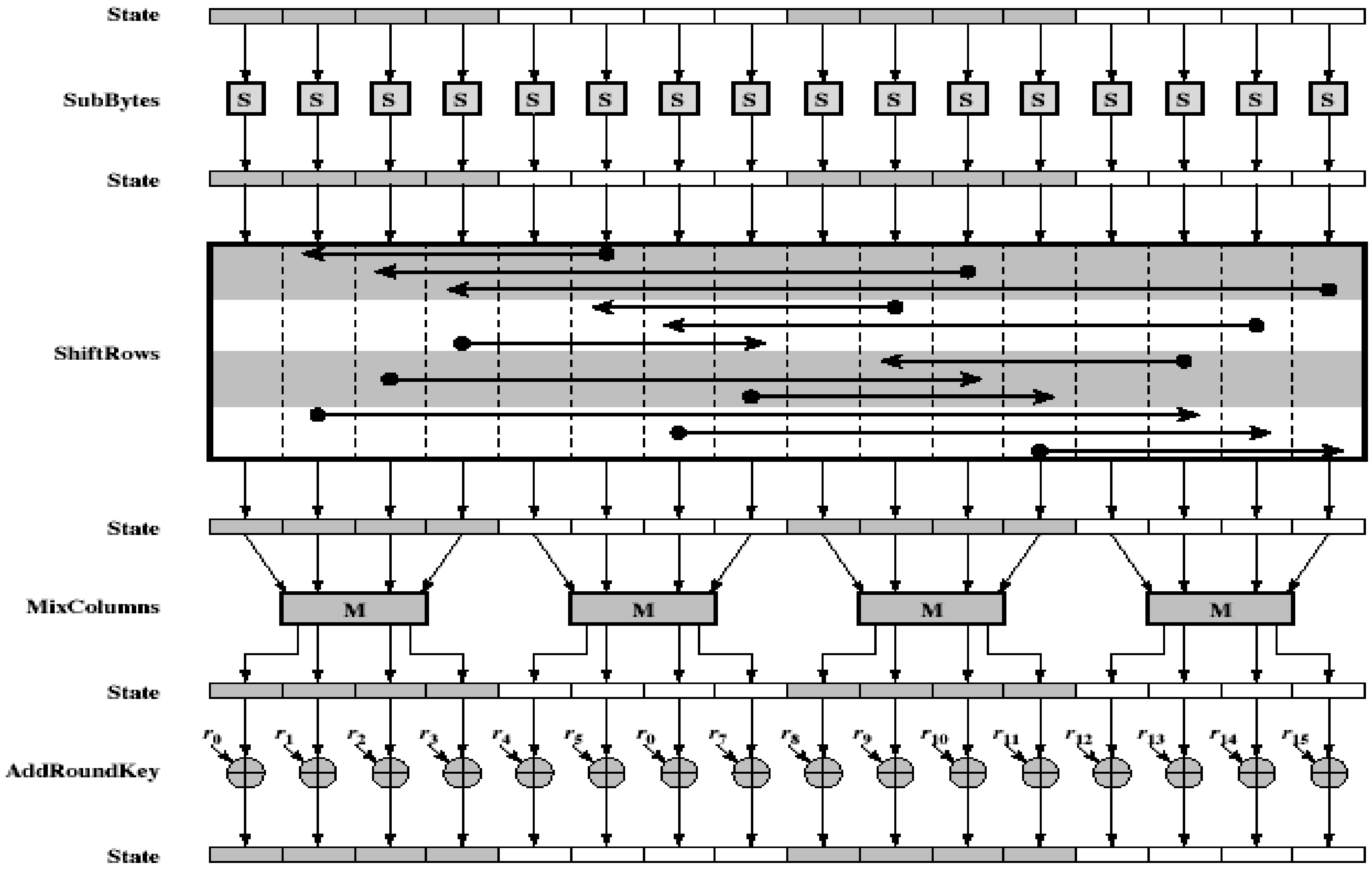
1b	22	cb	03	03	01	f1	23		
7c	ae	f4	ba	ab	22	ac	a3		
14	01	1b	4f	4c	03	02	39		
09	a6	88	4a	a5	88	22	39		

…

subkey generation

Key generated

AES Round



State
SubBytes
State
ShiftRows
State
MixColumns
State
AddRoundKey
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - every 4th has S-box + rotate + XOR constant of previous before XOR together
- designed to resist known attacks

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shifting
  - add round key works on byte XORs
  - mix columns requires matrix multiply in $GF(2^8)$ which works on byte values, can be simplified to use a table lookup

Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can pre-compute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 16Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Summary

- have considered:
  - the AES selection process
  - the details of Rijndael – the AES cipher
  - looked at the steps in each round
  - the key expansion
  - implementation aspects

brute force exhaustive search of AES-256

Computing power	Average time to crack using exhaustive search
High-end PC	27,337,893,038,406,611,194,430,009,974,922,940,323,611,067,429,756,962,487, 493,203 years. 27 trillion trillion trillion trillion trillion years
Fastest supercomputer	27,337,893,038,406,611,194,430,009,974,922,940,323,611,067,429,756,962,487 years. 27,337,893 trillion trillion trillion trillion years
2 billion high-end PCs	13,668,946,519,203,305,597,215,004,987,461,470,161,805,533,714,878,481 years 13,689 trillion trillion trillion trillion years
Age of the universe	15,000,000,000 years 15 billion years

https://scrambox.com/article/brute-force-aes/#:~:text=On%20average%2C%20to%20brute%2Dforce,simply%202255%20%2F%202%2C117.8%20trillion.

References

- William Stallings, Network Security Essentials : Applications and Standards, ISBN: 9788131761755, 8131761754

- Thanks to the many unknown sources from where some information is adopted.